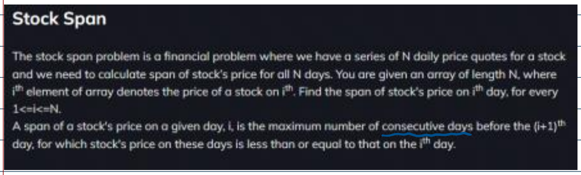
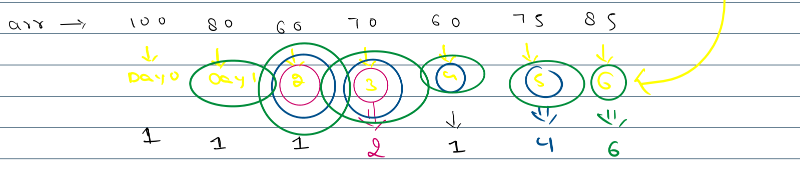
26 October 2025 10:14

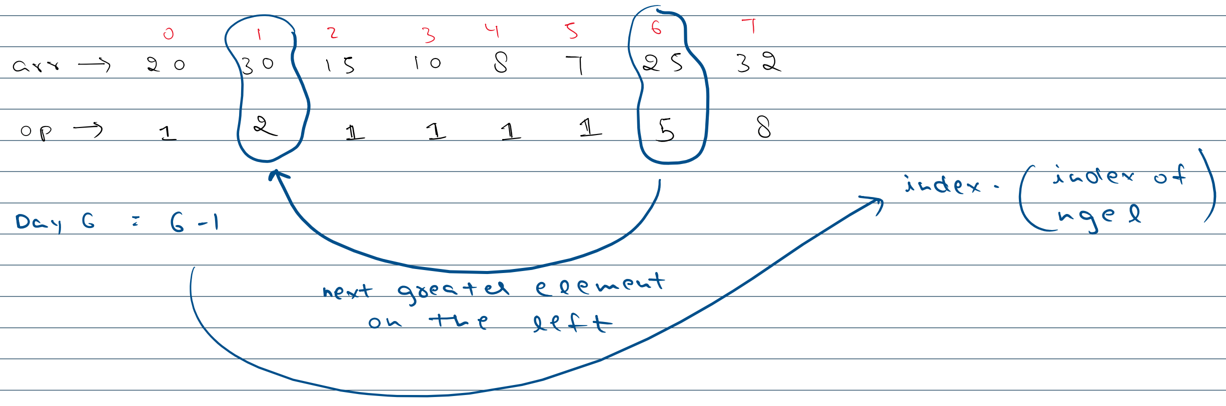


Stock Span

The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate span of stock's price for all N days. You are given an array of length N, where i^{th} element of array denotes the price of a stock on i^{th}. Find the span of stock's price on i^{th} day, for every 1<=i<=N.
A span of a stock's price on a given day, i, is the maximum number of consecutive days before the (i+1)th day, for which stock's price on these days is less than or equal to that on the i^{th} day.

Tata Steel





```
public class Main {
    public static void main(String[] args) {
        int[] arr = {20, 30, 15, 10, 8, 7, 25, 32};
        display(arr);
        int[] ans = ngel(arr);
        display(ans);
    }


    public static int[] ngel(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=0; i<arr.length; i++){
            if(st.isEmpty()){
               ans[i] = i - (-1);
            }
            else if(st.peek() > arr[i]){
                ans[i] = i - st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] <= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = i - (-1);
                }
                else{
                    ans[i] = i - st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }


    public static void display(int[] a){
        for(int i=0; i<a.length; i++){
```

```
            System.out.print(a[i] + " ");
        }
        System.out.println();
    }

}
```

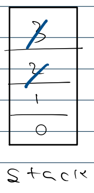
Celebrity Problem

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 1 & 1 & 1 & 1 \end{array}$$

a, b = 1 ⇒ $a \rightarrow b$ (a knows b)
a, b = 0 ⇒ $a \not\rightarrow b$ (a does not knows b)

row, col

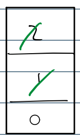
1, 2 ⇒ 1 knows 2
2, 1 ⇒ 2 does not know 1



pop → ~~3~~
pop → 2

3 knows 2 ⇒ 3 cannot be a celebrity
⇓
?

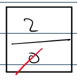
1



pop → 2
pop → ~~1~~

2 knows 1 ⇒ 2 can be a celebrity
⇒ 1 cannot be a celebrity
⇓
?

0



pop → 2
pop → 0

2 knows 0
⇓
⇒ 2 can be a celebrity
⇒ 0 can not be a celebrity

0

$\boxed{2}$

$\boxed{cel = 2}$

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 \\ 3 & 1 & 1 & 1 & 1 \end{array}$$

→ 0 0 0 0

→ exception

cel

2, 0
2, 1
2, 2
2, 3

→ 0

arr[cel][i] == 0

→ i == cel ✗

cel

0, 2
1, 2
2, 2
3, 2

→ 1

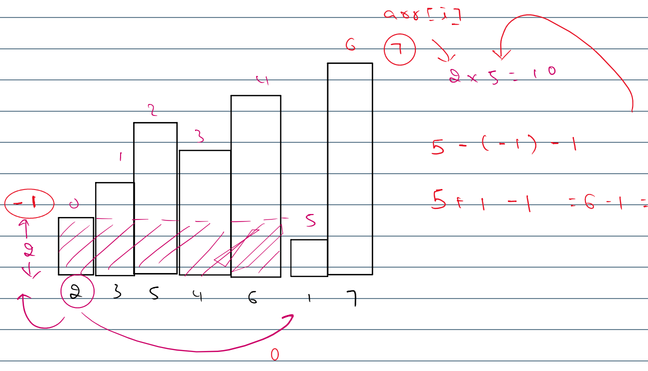
arr[i][cel] == 1

```
public class Main {
    public static void mai
        int[][] arr = {
            {1, 1, 1, 1},
            {1, 1, 1, 1},
            {0, 0, 1, 0},
            {1, 1, 1, 1}
        };

        System.out.printl
    }

    public static int celeb
        Stack<Integer> st
        for(int i=0; i<arr.l
            st.push(i);
        }

        while(st.size() > 1)
            int a = st.pop()
            int b = st.pop()
            if(arr[a][b] ==
                st.push(b);
            }
            else{
                st.push(a);
            }
        }

        int cel = st.pop();

        for(int i=0; i<arr.l
            if(i == cel){
                continue;
            }
            if(arr[cel][i] ==
                return -1;
            }
        }
        return cel;
    }

}
```

Largest Rectangle in Histogram

$$\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 2, & 3, & 5, & 4, & 6, & 1, & 7 \end{array}$$

[2, 3, 5, 4, 6, 1, 7]

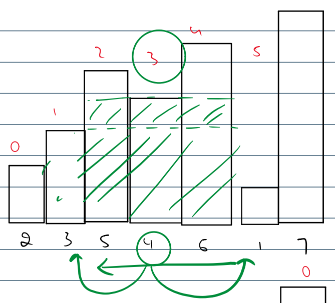


2 × 5 = 10

5 - (-1) - 1

5 + 1 - 1 = 6 - 1 = 5

```
n(String[] args) {






(celebrity(arr));


rity(int[][] arr){
= new Stack<>();
ength; i++){



{
;
;
1){









ength; i++){


 1 || arr[i][cel] == 0){
```



$\boxed{3 \times 4 = 12}$

arr[i]

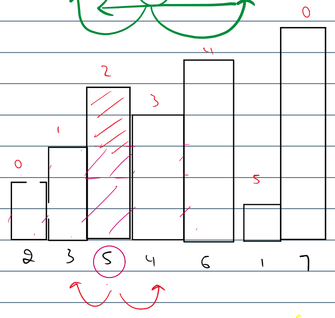
right[i] - left[i] - 1

5 - 1 - 1 = 3

nser = 5

nsel = 1

5 - 1 - 1 = 4 - 1 = 3



$\boxed{5 \times 1 = 5}$

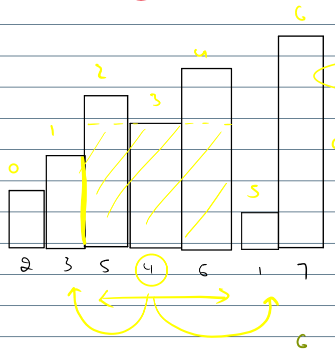
5 ↓ 2

→ 3 → 1 ⇒ nsel

→ 4 → 3 ⇒ nser

$\boxed{3 - 1 - 1 = 1}$

3 - 1 - 1 = 1



$\boxed{4 \times 3 = 12}$

arr[i]

5 - 1 - 1 = 3

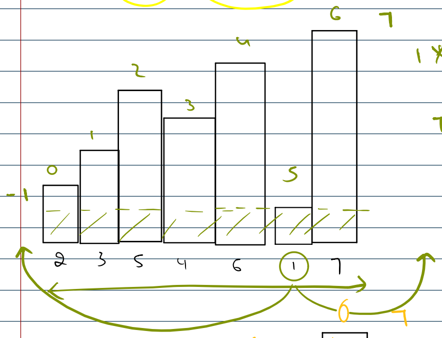
4 ↓ 3

→ 5 → 2 → nsel

→ 6 → 4 → nser

5 - 1 - 1 = 3

nser - nsel - 1

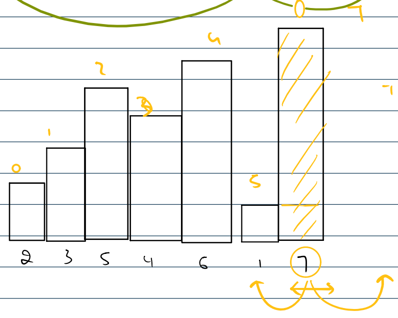


1 * 7 = 7

7 - (-1) - 1 = 7 + 1 - 1 = 7

nser → 7

nsel → -1



7 × 1 = 7

7 - 5 - 1 = 1

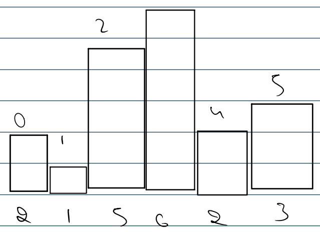
Area = (nsel - nser) * ~~width~~ 1 * height

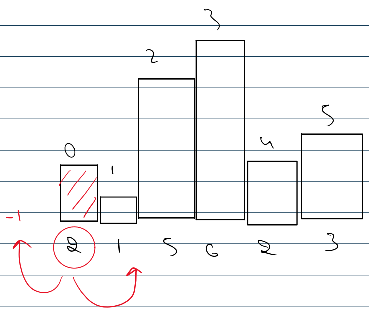
Area = (nsel - nser) * ~~heigth~~ → arr[i]

$\boxed{\text{width = right[i] - left[i] - 1}}$

[2,1,5,6,2,3]

3





nsel $\rightarrow$ -1

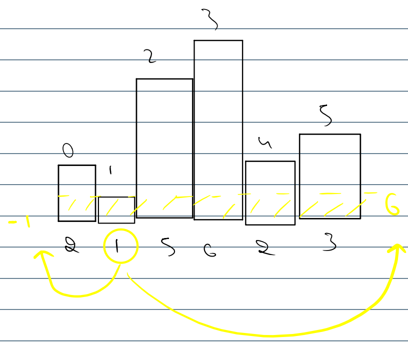
nser $\rightarrow$ 1

nser - nsel - 1

1 - (-1) - 1 = $\boxed{1}$

Area = 2 × 1

arr[i]



nser : 6

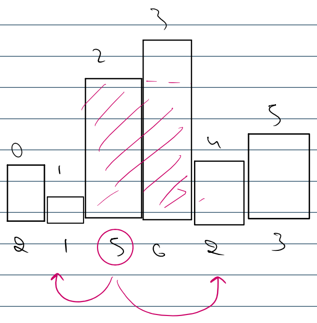
nsel : -1

nser - nsel - 1

6 - (-1) - 1 = 6

Area = 6 * 1 = 6

arr[i]



nsel : 1

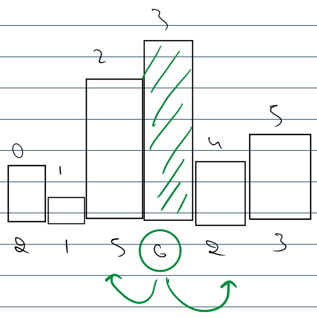
nser : 4

nser - nsel - 1

4 - 1 - 1 = $\boxed{2}$

Area = 2 × 5 = 10

arr[i]

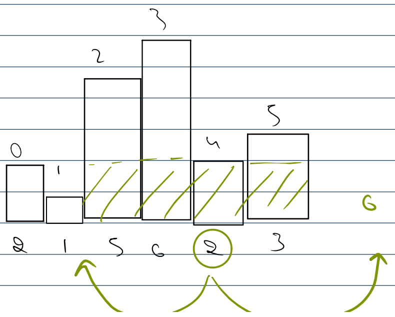


nser : 4

nsel : 2

4 - 2 - 1 = 1

Area = 1 * 6 = 6



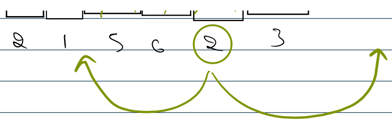
nser = 6

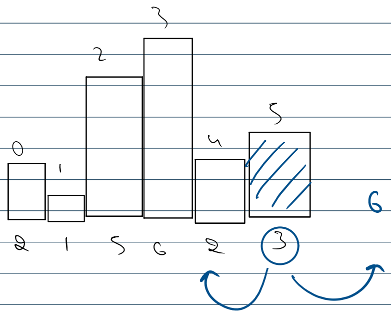
nsel : 1

6 - 1 - 1 = 4

Area = 4 * 2

arr[i]





nser : 6

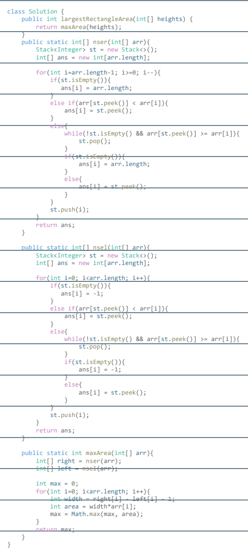
nsel : 4

6 - 4 - 1 = 1

Area : $\underbrace{1}_{} * \underbrace{3}_{}$ $\rightarrow$ arr[i]

= 3

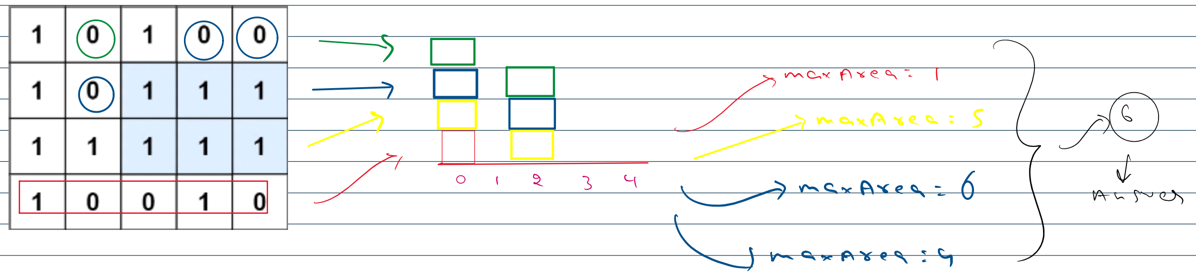
```
public class Main {
    public static void main(String[] args) {
        int[] arr = {2, 1, 5, 6, 2, 3};
        display(arr);
        System.out.println(maxArea(arr));
    }

    public static int[] nser(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=arr.length-1; i>=0; i--){
            if(st.isEmpty()){
               ans[i] = arr.length;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = arr.length;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=0; i<arr.length; i++){
            if(st.isEmpty()){
               ans[i] = -1;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
```

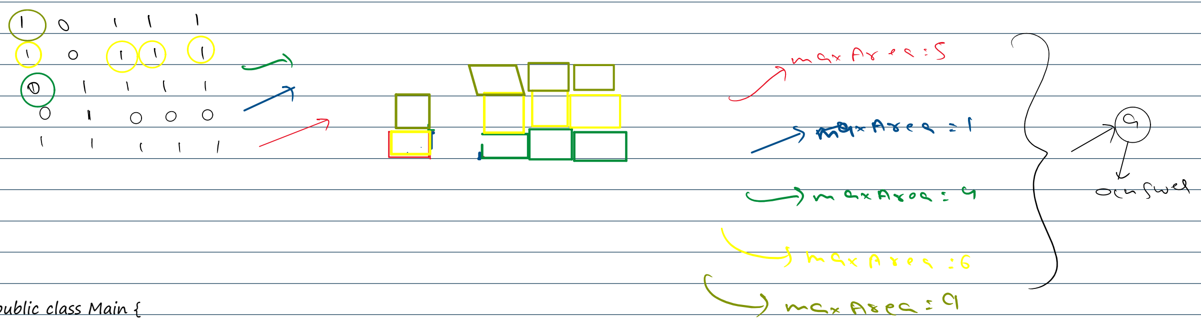


```
class Solution {
    public int largestRectangleArea(int[] heights) {
        return maxArea(heights);
    }
    public static int[] nser(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=arr.length-1; i>=0; i--){
            if(st.isEmpty()){
               ans[i] = arr.length;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = arr.length;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=0; i<arr.length; i++){
            if(st.isEmpty()){
               ans[i] = -1;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr){
        int[] right = nser(arr);
        int[] left = nsel(arr);

        int max = 0;
        for(int i=0; i<arr.length; i++){
            int width = right[i] - left[i] - 1;
            int area = width*arr[i];
            max = Math.max(max, area);
        }
        return max;
    }
}
```

```
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr){
        int[] right = nser(arr);
        int[] left = nsel(arr);

        display(left);
        display(right);

        int max = 0;
        for(int i=0; i<arr.length; i++){
            int width = right[i] - left[i] - 1;
            int area = width*arr[i];
            max = Math.max(max, area);
        }
        return max;
    }

    public static void display(int[] a){
        for(int i=0; i<a.length; i++){
            System.out.print(a[i] + " ");
        }
        System.out.println();
    }

}
```

maximal Rectangle





```
public class Main {
    public static void main(String[] args) {
        char[][] matrix = {
            {'1','0','1','0','0'},
            {'1','0','1','1','1'},
            {'1','1','1','1','1'},
            {'1','0','0','1','0'}
        };

        int[] arr = new int[matrix[0].length];
        int ans = 0;
        for(int i=0; i<matrix.length; i++){
            for(int j=0; j<matrix[0].length; j++){
                if(matrix[i][j] == '1'){
                    arr[j] = arr[j] + 1;
                }
```

```
                else{
                    arr[j] = 0;
                }
            }
            ans = Math.max(ans, maxArea(arr));
        }
        System.out.println(ans);
    }

    public static int[] nser(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=arr.length-1; i>=0; i--){
            if(st.isEmpty()){
               ans[i] = arr.length;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = arr.length;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=0; i<arr.length; i++){
            if(st.isEmpty()){
               ans[i] = -1;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr){
        int[] right = nser(arr);
        int[] left = nsel(arr);

        display(left);
        display(right);

        int max = 0;
        for(int i=0; i<arr.length; i++){
            int width = right[i] - left[i] - 1;
```

```
            int area = width*arr[i];
            max = Math.max(max, area);
        }
        return max;
    }

    public static void display(int[] a){
        // for(int i=0; i<a.length; i++){
        //      System.out.print(a[i] + " ");
        // }
        // System.out.println();
    }

}
```

```
class Solution {
    public int maximalRectangle(char[][] matrix) {
        int[] arr = new int[matrix[0].length];
        int ans = 0;
        for(int i=0; i<matrix.length; i++){
            for(int j=0; j<matrix[0].length; j++){
                if(matrix[i][j] == '1'){
                    arr[j] = arr[j] + 1;
                }
                else{
                    arr[j] = 0;
                }
            }
            ans = Math.max(ans, maxArea(arr));
        }
        return ans;
    }
    public static int[] nser(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=arr.length-1; i>=0; i--){
            if(st.isEmpty()){
               ans[i] = arr.length;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = arr.length;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int[] nsel(int[] arr){
        Stack<Integer> st = new Stack<>();
        int[] ans = new int[arr.length];

        for(int i=0; i<arr.length; i++){
            if(st.isEmpty()){
               ans[i] = -1;
            }
            else if(arr[st.peek()] < arr[i]){
                ans[i] = st.peek();
            }
            else{
                while(!st.isEmpty() && arr[st.peek()] >= arr[i]){
                    st.pop();
                }
                if(st.isEmpty()){
                    ans[i] = -1;
                }
                else{
                    ans[i] = st.peek();
                }
            }
            st.push(i);
        }
        return ans;
    }

    public static int maxArea(int[] arr){
        int[] right = nser(arr);
        int[] left = nsel(arr);

        int max = 0;
        for(int i=0; i<arr.length; i++){
            int width = right[i] - left[i] - 1;
            int area = width*arr[i];
            max = Math.max(max, area);
        }
        return max;
    }
}
```

```
class Solution {
    public int calPoints(String[] operations) {
        Stack<Integer> st = new Stack<>();
        int sum = 0;
        for(String op : operations){
            if(op.equals("C")){
                int removed = st.pop();
                sum = sum - removed;
            }
            else if(op.equals("D")){
                int add = st.peek()*2;
                st.push(add);
                sum = sum + add;
            }
            else if(op.equals("+")){
                int top = st.pop();
                int second = st.peek();
                int sum1 = top + second;
                st.push(top);
                st.push(sum1);
                sum = sum + sum1;
            }
            else{
                st.push(Integer.parseInt(op));
                sum = sum + Integer.parseInt(op);
            }
        }
        return sum;

    }
}
```